Wissenschaft vs. Design: Konstruktionslehren für den Maschinenbau, den Computer und die Software im historischen Diskursvergleich

Hellige, Hans Dieter

Veröffentlichungsversion / Published Version
Konferenzbeitrag / conference paper

Zur Verfügung gestellt in Kooperation mit / provided in cooperation with:
SSG Sozialwissenschaften, USB Köln

artec Forschungszentrum Nachhaltigkeit

Hans Dieter Hellige

Wissenschaft vs. *Design*: Konstruktionslehren für den Maschinenbau, den Computer und die Software im historischen Diskursvergleich

artec-paper Nr. 153
März 2008

Beitrag für den Workshop »Wissenschaft oder Design« auf der Tagung des Fachbereichs Informatik und Gesellschaft der GI "Kontrolle durch Transparenz – Transparenz durch Kontrolle", Berlin-Adlershof, 27.-29- April 2007

ISSN 1613-4907

Universität Bremen

artec - Forschungszentrum Nachhaltigkeit
Enrique-Schmidt-Str. 7
Postfach 330 440
28334 Bremen
http://www.artec.uni-bremen.de

WISSENSCHAFT VS *DESIGN*: KONSTRUKTIONSLEHREN FÜR DEN MASCHINENBAU, DEN COMPUTER UND DIE SOFTWARE IM HISTORISCHEN DISKURSVERGLEICH

Hans Dieter Hellige

Seit langem ringen in den Konstruktionsauffassungen und Konstruktionslehren verschiedener Technikbereiche künstlerisches Designverständnis auf der einen und Verwissenschaftlichungs- bzw. Rationalisierungs-Leitbilder auf der anderen Seite miteinander. In Frühphasen neuer Technikbereiche dominieren meist Apotheosen der letztlich 'unberechenbaren Ingenieurkunst', während mit zunehmender Etablierung einer Technik immer mehr Empirisierung, Theoretisierung, Systematisierung und Berechenbarkeit in den Vordergrund treten. Doch das klare Entwicklungsmodell eines kontinuierlichen Verwissenschaftlichungsprozesses des Konstruktionswissens nach dem Muster klassischer Naturwissenschaften wird durch mehrfache Moden- und Methodenwechsel sowie durch sich immer wieder erneuernde Kontroversen zwischen den Wissenschaftskulturen der empirischen Praxis und der szientistischen Academia gestört. Für den Historiker ergibt sich dadurch bei der Rekonstruktion der Entwicklung von Konstruktionslehren das Problem der theoretischen Grundlegung. Soll er trotz gegenläufiger Entwicklungsphasen doch letztlich das szientistische Modell der Wissenschaftsgenese zugrunde legen und den Entwicklungsgang teleologisch auf das Ziel strenger Wissenschaftlichkeit nach naturwissenschaftlichem Muster ausrichten? Oder soll er sich mit der Position des Konstrukteurs als Künstler und seinem Protest gegen die Rationalisierung des geistigen Schaffensprozesses identifizieren und damit Abschied von einem szientistischen Fortschrittskonzept nehmen? Oder soll er von einem spiralförmigen Lernprozess ausgehen, der die konträren Entwicklungsmodelle harmonisiert? Im Folgenden möchte ich demgegenüber für eine offene Rekonstruktion der Diskursprozesse und Wissenschaftskontroversen plädieren, die dem Entwicklungsgang und ihren Lernprozessen nicht von vornherein eine bestimmte Entwicklungslogik unterstellt. Als methodisch besonders ergiebig erweist sich dabei ein Diskursvergleich von Konstruktionslehren unterschiedlicher Technikbereiche, da sich hierbei interessante Übereinstimmungen und Abweichungen ergeben, die Erkenntnisse über die jeweiligen Disziplingrenzen hinaus ermöglichen.

Denn in dem Wechselspiel von auf Erfahrung beruhenden Gestaltungsansätzen und den Versuchen einer Verwissenschaftlichung in der 150-jährigen Geschichte der Konstruktionslehre im Maschinenbau zeigt sich, dass wenn ein Mangel an qualifizierten Konstrukteuren auftrat, der Ruf nach einer systematisch lehrbaren Konstruktionsmethodik laut wurde. Und immer stellte sich bald heraus, dass diese ›wissenschaftlichen‹ Methoden, die eventuell zur funktionalen Analyse von Bauteilen hilfreich waren, nicht für ihre Konstruktion ausreichten,

weil dabei wichtige Designaspekte wie Material, Kosten etc. ausgeblendet wurden und der unkontrollierbaren Erfahrung der Konstrukteure überlassen werden mussten. Die Parallelen zu den Diskussionen in der viel kürzeren Geschichte der Computerarchitektur und der Softwareerstellung sind frappierend, und die ›ehrwürdige‹ Geschichte der Konstruktionslehre stellt so einen guten Hintergrund dar, die Auseinandersetzungen in der Computerarchitektur und im Software-Engineering zu beurteilen. [1]

1 DIE KONSTRUKTIONSLEHREN VON MECHANIK UND MASCHINENBAU

Bei der Durchsicht von Schlüsseltexten und Fachzeitschriften der Konstruktionslehre und -forschung in Deutschland ergibt sich ein charakteristischer Wandel der Analogiemodelle, Metaphern und Leitbilder für das Konstruieren:

In der Anfangszeit wurde der Konstruktionsprozess vor allem mit dem *künstlerischen Schaffensprozess* und dem *philosophischen Erkenntnisprozess* verglichen, meist mit einem berufsständischen Impetus. Unter dem Einfluß der Reuleaux'schen Kinematik wurde dann das *Elemente-Montage-Modell* mit den Prinzipien der Elementarisierung und des Austauschbaus auf die klassische Maschinenbaulehre übertragen und später sogar auf die Elementarisierung der Denkvorgänge ausgedehnt. Die zunehmende Bedeutung der Serienfabrikation führte seit der Jahrhundertwende zu einer Annäherung des Prozessmodells der Konstruktion an das der Fertigung. Dies gipfelte nach dem Ersten Weltkrieg im Leitbild des *Fließmodells,* das den Konstruktionsprozess analog zur Fließbandarbeit sieht. In den 30er und 40er Jahren ergab sich unter dem Einfluss von Elektromechanik und Feinwerktechnik durch die Verknüpfung des Fließbild- und Elemente-Montage-Modells das *Schaltplan-Modell,* daneben auch das Vorbild des mathematischen Lösungsverfahrens und das Feldmodell der Konstruktionsentscheidungen. Dadurch wurde die Konstruktionsmethodik als eine systematische Konstruktions'technik' angelegt. Nach dem Zweiten Weltkrieg entstanden durch Anleihen an die Regelungstechnik bzw. Kybernetik, Systemforschung und Datenverarbeitung aus dem Schaltplan-Modell das

---

1 Der Beitrag beruht wesentlich auf den folgenden Studien, die auch die Quellen- und Literaturbelege bringen: Hellige, Hans Dieter: Hierarchische Ablaufsteuerung oder kooperative Bewältigung von Problemzusammenhängen? Zur Geschichte von Modellen des Konstruktionsprozesses, in: Hellmuth Lange, Wilfried Müller (Hrsg.), Kooperation in der Arbeits- und Technikgestaltung, Münster, Hamburg 1995, S. 135-164; Hellige, Hans Dieter (2003): Zur Genese des informatischen Programmbegriffs: Begriffsbildung, metaphorische Prozesse, Leitbilder und professionelle Kulturen, in: Karl-Heinz Rödiger (Hrsg.), Algorithmik - Kunst – Semiotik, Heidelberg 2003, S. 42-73; Hellige, Hans Dieter: Die Genese von Wissenschaftskonzepten der Computerarchitektur: Vom „system of organs“ zum Schichtenmodell des Designraums, in: Hellige, Hans Dieter (Hrsg.), Geschichten der Informatik. Visionen, Paradigmen und Leitmotive, Berlin, Heidelberg, New York 2004, S. 411-471; Hellige, Hans Dieter, Software-Manufactory - Software-Architecture - Software-Engineering: Conceptual Change in Software-Construction in 1950s and 60s, 35 S., erscheint voraussichtlich in: IEEE Annals of the History of Computing, 2009/2

kybernetische, das *systemtechnische Modell* und das *Modell des informationsverarbeitenden Prozesses*, die bis heute die verschiedenen Richtungen der deutschen und westeuropäischen Konstruktionsmethodik bestimmen.

Mit diesen Theorieanleihen wurde gezielt auf eine Verwissenschaftlichung des Konstruktionsprozesses hingearbeitet. Die Leitbilder der "Konstruktion auf Knopfdruck", des "Konstruktionsalgorithmus" und des "gespeicherten Konstruktionswissens auf Abruf" führten zu einer Reduktion der komplizierten Problemlösungs- und Gestaltungsprozesse auf ein tendenziell algorithmisches Elemente-Auswahl- und -Montageverfahren aus kompletten Lösungs- und Objektspeichern bzw. aus einem hierarchischen Baukastensystem. Die konstruktive Problembearbeitung wurde ungeachtet aller Beteuerungen ihres schöpferischen Charakters aus der Modellsicht des modularisierten, auf Wiederholelemente getrimmten Konstruktionsgegenstandes gesehen, der auf jeder Abstraktionsstufe (System-, Funktions- und Gestaltelemente) die gleiche regelmäßige Blockstruktur aufweist. Die durchweg kybernetische und informationstechnische Modellierung schuf eine enge Strukturanalogie von Maschine/Produkt, Herstellungs- und Konstruktionsprozess und zerstörte damit die analytische Differenz, die zum Begreifen der jeweiligen Wesensunterschiede und abweichenden Gestaltungsspielräume erforderlich ist.

Die anfangs bewusste, später halb- und unbewusste Übertragung technischer Strukturen und Modelle auf Denk- und Arbeitsabläufe suggerierte die mögliche Gleichbehandlung von Mensch und Maschine. Sie ebnete außerdem den heterogenen, disparaten und kontingenten Charakter des sich im Rahmen des Alltags- und Gesellschaftsbewusstseins vollziehenden Arbeitshandelns ein, indem die allgegenwärtigen Regelkreise, Blockschaltbilder und Informationsflussdiagramme die Problementscheidungen der Konstrukteure zum bloßen Schaltelement bzw. zur Durchflussgröße einer logistischen Kette verkürzten. In Gestalt der technischen Modellierung verselbständigten sich die zugrunde liegenden Rationalisierungsleitbilder und drängten dadurch die anderen Aspekte des methodischen Konstruierens zurück, so vor allem die Kreativitätsförderung, die Ausweitung von Lösungsspektren und Sichtbarmachung von Gestaltungsspielräumen, eine stärkere Gebrauchswert- und Benutzerorientierung sowie die Einbeziehung der Langzeitperspektive von Produkten und Techniken. Insgesamt haben dominante Phasenabschottung, einseitige CAD-Fixierung und die durchgängige systemtechnisch-kybernetische Modellierung von Konstruktionsobjekt, -subjekt und -prozess das Kontingenzbewusstsein und die Gestaltungsorientierung eher gehemmt als gefördert.

Es ist offensichtlich, dass die erwähnten Analogiemuster den jeweils erreichten Stand der Produktivkräfte auf den Konstruktionsprozess projizieren und damit auch entsprechende Rationalisierungskonzepte auf ihn übertragen, ohne diese bisher wirklich durchsetzen zu können. In der Gegenwart wie in der Vergangenheit scheiterten die Versuche über die fertigungs-,

regelungs- und informationstechnischen Modellierungen der Elemente-Montage, des Fließbandes, des Schaltplanes, des Regelkreises und des Algorithmus prozesstrukturierend auf die gesamte Konstruktionsarbeit einzuwirken, nämlich meist an deren komplexen Charakter. Es gelang nicht, mit den technischen Modellen auch die komplizierten Aufgaben-, Problem- und Entscheidungsstrukturen realer kooperativer Gestaltungsprozesse abzubilden. Es entstanden daher zunächst oft als Gegenmodelle, dann auch als Nebenmodelle zu den Prozessketten, deskriptive Problemaufrisse bzw. später mehr oder weniger elaborierte Problemstrukturpläne, die die strategischen Festlegungen von Aufgaben oder Produktstrukturen bzw. Produkteigenschaften in Wechselbeziehung zu den Anforderungen und dem gesellschaftlichen wie ökologischen Konstruktionsumfeld thematisierten. Doch standen in der Geschichte der Konstruktionswissenschaft als Folge vorherrschender Rationalisierungsintentionen die Aufgabenpläne fast immer im Windschatten der Ablaufschemata.

Die vorhandenen Ansätze zur Problemstrukturierung und Entscheidungskonflikt-Bewältigung sind aber als Anknüpfungspunkte für die Konstruktionsforschung von großem Interesse, da in ihnen implizit der Prozess der Umsetzung von gesellschaftlichen Anforderungen und Wertkriterien in Designentscheidungen behandelt wird. Anreize zu einer stärkeren Berücksichtigung von Problemstrukturen und Kooperationsproblemen kommen neuerdings von dem Wandel in den Produktionskonzepten. Durch das Ziel, über ein sog. Simultaneous bzw. Concurrent Engineering die sequentiellen Arbeitsschritte zu parallelisieren, um so den Konstruktionsprozess drastisch zu beschleunigen, wird erstmals auch das quasi-tayloristische bzw. fordistische Phasenmodell als zentraler Strukturierungsansatz der Konstruktionsmethodik in Frage gestellt. Dabei kündigt sich ein Wechsel von der sequentiellen zur Parallelverarbeitung an. In der letzten Zeit gibt es als Folge von Vernetzung, objektorientierten Softwaretechniken und Gruppenarbeitskonzepten erste Ansätze für *Netzmodelle und Objektmodelle* des Konstruktionsprozesses. Im letzten Jahrzehnt hat man zudem den Konstrukteur und sein empirisches Konstruktionsverhalten als Forschungsthema entdeckt, wobei man zunehmend unter den Einfluss postmoderner Designtheorien geriet, die die subjektiven Aspekte und den 'künstlerischen' Charakter der Konstruktion wieder deutlich herausheben.

2 DIE KONSTRUKTIONSLEHREN IN DER COMPUTERARCHITEKTUR

Auch in der "Computerarchitektur" stehen am Beginn *Kompositionslehren*, die der Designkomplexität voll Rechnung tragen und nicht selten das Konstruieren als "Kunst" bzw. "art" verstehen. Aus ihnen entwickelten sich mit zunehmender Reife *Dekompositionslehren*, die vorrangig auf Standardisierung, Prozessrationalisierung und Reduktion von Designkomplexität zielen. Systematische Reflexionen über Systemstrukturen und Struktur- bzw. Bauform-Variationen, Stilbildungen und Entwurfsmethoden werden dabei in der Computer-Konstruktion seit 1961 mit der Architektur-Metapher belegt. Die eigentlichen Anfänge des theoretisch-

methodischen Architekturkonzeptes bei Computersystemen liegen jedoch schon lange vor der Einführung des ästhetisierenden Architekturbegriffs durch Frederick Brooks. Die frühesten architektonischen Designreflexionen finden sich bereits bei einigen Computerpionieren kurz vor und während des Zweiten Weltkrieges. Diese behandelten noch vor dem eigentlichen Bau bzw. kurz nach Fertigstellung der eigenen Rechenanlage in grundsätzlicher Weise Aufgabenbereiche, 'Organ'-Struktur und Ablaufprozeduren eines Computers in ihrem inneren Zusammenhang. Seit den frühen 50er Jahren stützen sich 'Architektur'-Erwägungen dann auf Auswertungen der Erfahrungen mit den Pionierentwicklungen sowie auf erste Vergleiche der verschiedenen Bauweisen, Bauformen und Designprinzipien. Mit einer integrativen Betrachtung der Designmethoden und einer Systematisierung der Bauformen erlebte das Architekturkonzept dann in den 60er und 70er Jahren seinen Durchbruch im Computerbereich. Mit der Methode der System-(De)Komposition förderte man die Bildung von Modulhierarchien und Ebenenmodellen, die den Entwicklungsgegenstand arbeitsteilungsgerecht modellierten und so die Grundlage für elaborierte Prozessketten werden konnten. Durch Annäherung an die Modul- und Synthesetechnik der Elektro- bzw. Elektronikkonstruktion sowie an systemtechnische Methoden der System-, Produkt- und Prozessmodellierung schien das Computerdesign weitgehend zu einer lehr- und lernbaren Konstruktionstechnik zu werden, die aus vorhandenen Bausätzen und Lösungsmustern anforderungsgemäße Anlagen konfiguriert und montiert. Doch entsprach diese an etablierten Technikzweigen orientierte Entwicklungs- und Konstruktionsmethodik noch nicht dem damaligen Stand der Computertechnik. Trotz der zunehmenden Durchsetzung der ”klassischen von Neumann-Architektur” war die Entwicklung noch nicht so weit abgeschlossen, waren die Bauformen noch nicht so standardisiert und die Designmethoden noch nicht so ausgereift, dass ein am Austauschbau und Fertigungsprozessen ausgerichtetes Elemente-Montage-Konzept das adäquate Muster für die Technikwissenschaft und Designlehre des Computers darstellte.

Anfang der 60er Jahre entstand daher das "Architektur"konzept, das die Betrachtung gerade auf die bisher unterbelichteten Syntheseaspekte und technisch-organisatorischen Zusammenhänge des Designs und der Struktur eines Computersystems fokussierte. Die Perspektive verschob sich hier vom Best-way-Prinzip, von der Elementarisierung und Bausatz-Montage zur Organisation der Gesamtstruktur und zum erfahrungsbasierten Management der Abstimmungs-, Integrations- und Syntheseprozesse hin. Nicht mehr die möglichst rationelle Entwicklung und Konstruktion war das vorrangige Ziel, sondern die nutzer- bzw. nutzungsbezogene Produktoptimierung. Als Leitbegriff für diesen Perspektivwechsel und erweiterten Ansatz der Komplexitätsbewältigung im Rechnerdesign wurde um 1960 nun der Begriff "architecture" in die Computer Community eingeführt. Dieser zunächst auf die technisch-organisatorische Integrations- und Vermittlungsfunktion des Designers ausgerichtete Architekturbegriff wird Ausgangspunkt aller späteren design-orientierten Architekturkonzepte. Doch erst seit Beginn der 70er Jahre wird die sich herausbildende Computer-Designmethodik und –Technik-

wissenschaft endgültig mit Signum der "computer architecture" verbunden. Sie gerät dabei zunehmend unter den Einfluss von Strukturierungs- bzw. Modularisierungskonzepten und Engineering-Methoden, die seit dem Ende der 50er und verstärkt seit den 60er Jahren im Softwarebereich, speziell in den Betriebssystemen entstanden sind.

Der Architekturbegriff bündelte während der 60er Jahre eine ganze Reihe von Trends im Rechnerentwurf und in der Designer-Community. Er kennzeichnete die gewachsene Komplexität des Gesamtsystems von Hardware und Software und die daraus resultierenden vermehrten Lösungsalternativen und Designkonflikte. Er apostrophierte die integrativen Synthese-Leistungen in der Entwicklung, für die noch keine Ingenieurprinzipien und theoriebasierten Dekompositionsstrategien zur Verfügung standen. Dazu gehörten vor allem die Gestaltung der Gesamtfunktion und des Mensch-Computer-Interface sowie die Austarierung heterogener Anforderungen zu einem konsistenten Gesamtentwurf. Die Lösung des Syntheseproblems wie die intendierte hohe nutzeradäquate Designqualität wurde aber in erster Linie von einer aristokratischen Designkultur und einer hierarchischen Organisation des Entwicklungsprozesses erwartet, damit aber auch, ähnlich wie beim ebenfalls bei IBM entstandenen Chief-Programmer-Ansatz in der Softwarekonstruktion, von den Integrationsleistungen und der Designmanagement-Qualifikation der ”Chief-Designer” abhängig gemacht. Mit dieser Personalisierung und Ästhetisierung des Designproblems ließ sich zwar ein Professionalisierungsanspruch begründen, nicht aber die Konstituierung einer technischen Disziplin. Es entwickelte sich daher um 1970 eine Richtung in der ”architectural community”, die zwar am ”architectural model” als Grundlage des Designmanagements festhielt, doch die sich zugleich gegen eine zu starke Betonung des Kunstcharakters des Designs und gegen das überhöhte Leitbild des Allround-Systemarchitekten verwahrte. Gegenüber der Hoch- bzw. Überschätzung qualitativer Aspekte setzte man hier mehr auf quantitative Methoden, auf eine systematische Erfassung und Strukturierung des Designraums, die Entwicklung entsprechender Notationen und Werkzeuge sowie besonders auf die Deskription von Designkonflikten.

In den Architekturkonzepten der 70er bis 90er Jahre verschob sich mit fortschreitender Etablierung und Akademisierung der Disziplin sowie mit der wachsenden Komplexität der Computersysteme die Zielrichtung von der *Ausweitung* zur *Begrenzung* des Lösungsraumes. Ebenso wurde die bisherige Vielfalt der Modellsichten zugunsten weitgehend homogener Betrachtungsweisen und Repräsentationsformen reduziert, um so der Theoriebildung näher zu kommen. Die Designkonflikt-Modellierung wurde nun aufgegeben und durch Lösungsstrategien ersetzt, die die Komplexität und Heterogenität vorab ausschalteten. Dabei geriet die Computerarchitektur zunehmend unter den konzeptionellen Einfluss des Software-Engineering, wo die Komplexitäts-mindernden Strukturierungsmethoden schon seit den 60er Jahre vorherrschend waren. So kam es nun auch zu Ansätzen für eine "structured computer architecture", in denen das hierarchische Schichten- bzw. Modularisierungskonzept ganz in den

Mittelpunkt der Architekturlehre rückte. Eine andere Gruppe von Architekturlehren bilden die in den 80er und 90er Jahren aufkommenden theoriebasierten Taxonomien, die die Computerarchitektur durch enge Anlehnung an Naturwissenschaften wie die Chemie oder an die Mathematik endgültig zu einer strengen Technikwissenschaft weiterentwickeln möchten (u.a. Dasgupta; Everling). Die von Flynn (1972) begonnene, von Händler (1978), Skillikorn (1988) u.a weitergeführten Klassifikationsschemata werden dabei so fein gegliedert und theoretisch untermauert, dass sie ihren Charakter als historisch entwickelten Lösungsraum abstreifen und zu einer Art chemischer Notation aufsteigen, die alle möglichen Elementeverbindungen von der Atom- bis zur Makromolekülebene über Formeln zugänglich macht. Die Verwissenschaftlichung soll hier vor allem helfen, das Design von Zufallsentscheidungen und individuellen Konstruktionserfahrungen unabhängig zu machen. Dazu dienen besonders formale Beschreibungssprachen und strikte Klassifikationsprinzipien für die *exakte* Architekturdeskription und für eine automatische Exploration des "design space". Eine regelbasierte Konstruktionsmethodik mit vorab definierten Methodenschritten innerhalb eines voll durchsystematisierten Lösungsraumes soll hier den bisherigen "informal design process" weitestgehend ablösen. Der Entwicklungsgang von der Komplexität kunstvoll integrierenden Kompositionslehre zur Theorie-fundierten Dekompositionslehre schien damit zu seinem Abschluss gekommen zu sein.

Das besonders in den 70er/80er Jahren vertretene szientistische Paradigma und die exakten Dekompositionslehren mit dem Ideal der Rechner-kontrollierten Elemente-Montage-Konstruktion stießen aber offen-sichtlich an Grenzen. So wurden sie in den 90er Jahren wieder von empirischen Engineering-Ansätzen und einer Renaissance designbetonter Architekturkonzepte zurückgedrängt. Diese Entwicklungstendenzen in der Computerarchitektur scheinen zu signalisieren, dass sich angesichts der Fortdauer der technischen Umwälzungen und der stets noch wachsenden Nutzungs- bzw. Umgebungsbedingten Komplexität szientistische Methoden der Erfahrungskompression und Designrationalisierung nur begrenzt bewährt haben und dass sich die Disziplin damit wieder mehr als eine konstruktive Technikwissenschaft begreift. Deren Arbeitsgegenstand unterliegt nachwievor divergierenden Nutzer- bzw. Stakeholder-Anforderungen und kann daher als Ganzes nicht nach Vorbild der Mathematik und Naturwissenschaften streng wissenschaftlich modelliert werden, sondern muss nach dem Muster designintensiver Ingenieurdisziplinen mit einer Vielfalt von Modellsichten und der Heterogenität von Designkonflikten zurechtkommen.

3 DIE KONSTRUKTIONSLEHREN IN DER SOFTWARETECHNIK

Auch in der Softwarekonstruktion kann man einen mehrfachen Wechsel von stärker designorientierten und szientistischen Konstruktionslehren beobachten, was sich allein schon in den vorherrschenden Leitbegriffen und Disziplin-Bezeichnungen ablesen lässt. In den Software-

Konstruktionslehren dominierte jahrzehntelang der Begriff "Software-Engineering". während "architecture" keine allzu zentrale Rolle spielte. Der Architekturbegriff wurde gerade von den Computer Scientists gemieden, die den Hauptanteil an der Entwicklung von Strukturierungs- und Organisationsprinzipien für hochkomplexe Softwaregebilde hatten. Erst seit der Mitte der 80er Jahre gewann der Architekturbegriff an Gewicht gegenüber der seit 1968/69 die Fachdiskussion beherrschenden Leitmetapher "Software-Engineering" und erst ab 1992/93 begann auch hier die Herausbildung einer eigenständigen Disziplin "Software-Architektur", die mithilfe einer stärkeren Betonung von Stil und Gestaltungsaspekten das vorrangig fertigungsorientierte Industrialisierungs-Paradigma des Software-Engineering ablösen bzw. erweitern möchte.

Das Ideenreservoir des Software-Engineering und der Software-Architektur reicht jedoch schon zurück in die Anfänge des modernen Computers: John v. Neumann entwickelte bereits 1946/47 ein Phasenmodell des Programming und ging dabei, ähnlich wie gleichzeitig Alan Turing, von einem arbeitsteiligen Entwicklungsprozess aus. George R. Stibitz modellierte schon 1947/48 die Programm- und Rechnerabläufe als ein hierarchisches Ebenenmodell und versprach sich davon eine Erleichterung arbeitsteiliger Entwicklungs-, Test- und Änderungsprozesse. Und Alwin Walther stellte 1946/52 wohl als erster eine direkte Parallele zwischen der Programmherstellung und der Planung eines Fabrikationsprozesses her. Über derartige rein gedankliche Analogiebildungen gingen Wilkes, Wheeler und Gill mit ihrem Konzept einer Programm-Montage aus Standard-Subroutinen hinaus. Sie übertrugen 1951 die "modular design philosophy" von der Elektronik- Konstruktion auf die Software-Herstellung. Einen wirklichen Durchbruch erlebten Engineering-Konzepte im Softwarebereich aber erst Mitte der 50er Jahre im Rahmen der von Software Contractors entwickelten großen Softwaresysteme im Militär- und Luftfahrt-Bereich. Um den Umfang und die Komplexität der Aufgabe zu bewältigen, griff Herbert D. Bennington im SAGE-Projekt auf das in der Hardware-Entwicklung eingesetzte Methodeninstrumentarium des Engineering und Projektmanagement zurück. Hinzu kamen tayloristische Instrumente des "Industrial Engineering" wie Flowcharts und Gantt Diagrams zur Verfolgung der definierten "milestones". Aus diesem Methoden-Mix ging das erste Vorgehensmodell für den Software-Entwicklungsprozess von Larg-scale Software-Systemen hervor.

Doch im Laufe des Systementwicklung wurden Grenzen und Probleme des gewählten Prozessmodells immer deutlicher. So ließ sich die ursprünglich intendierte strikt arbeitsteilige Entwicklung von "decentralized programs" bei einem Real-Time-System mit hoher Inter-

dependenz der Einzelprozesse nicht aufrechterhalten. Die Software-Entwicklungsmethoden in der SAGE-Nachfolge versuchten deshalb zwar der Komplexitätssteigerung bei Online-, Real-Time- und Time-Sharing-Systemen weiterhin mit Phasenmodellen, Meilensteinen und der Aufteilung der Riesenprogramme in arbeitsteilig zu bearbeitende "blocks" oder "modules" zu begegnen. Doch ging man bei der Strukturierung von Produkt und Prozess meist nicht so strikt nach dem Muster des Industrial Engineering vor. Auch die ab 1960 von Projektmanagern großer militärischer Softwaresysteme formulierten Software-Entwicklungs-Lehren verstanden Phasenmodelle in der Regel nur als Leitlinien für die Programm-Entwicklung, an die man sich im konkreten Entwicklungsablauf nicht sklavisch halten dürfe. Die Strukturierungskonzepte und Managementmethoden für die Softwareherstellung dieser Zeit erscheinen so aufgrund der Vorstellung eines quasi-industriellen Fertigungs- und Montageprozesses von Komponenten bzw. Modulen nur auf den ersten Blick als eine Vorwegnahme der Software-Engineering-Ansätze am Ende der 60 Jahre. Doch bei näherem Hinsehen werden charakteristische Unterschiede deutlich:

- Die Strukturanalogie zum Austauschbau wird nur angedeutet, die interdependente Aufgaben- und Problemstruktur bleibt trotz der Modularisierungsbestrebungen erhalten.
- Das Phasenmodell zielt nicht auf eine strikt 'tayloristische' Arbeitsteilung, sondern bleibt eingebettet in ein aus heutiger Sicht modern erscheinendes Konzept des Concurrent Software-Development und des "evolutionary approach", in dem die Programmsysteme in enger Abstimmung mit den Benutzern evolutionär entstehen.
- Die Programmerstellung bleibt letztlich "art", ein teamartiger Designprozess, in dem die Real-Time-Programmer "creative work in a highly specialized environment" leisten.

Im Jahrzehnt vor den NATO-Konferenzen von 1968/69 waren im Bereich der militärischen Programm- und Systementwicklung großer Software- und Informationssysteme eine Reihe von Design-, Projekt- und Entwicklungsmanagement-Methoden entstanden, die schon in vielem die späteren evolutionären Ansätze von Meir M. Lehmann und die 'ganzheitlichen', auf Teamarbeit, Entwickler-Kreativität und Benutzerorientierung ausgerichteten Methoden der partizipativen und agilen Software-Entwicklung vorwegnahmen. Sie alle waren, obwohl weitgehend aus militärischer Umgebung hervorgegangen, betont teamorientiert, sie standen in der Tradition der interdisziplinären Expertenteams des New Deal und der fach- und professionsübergreifenden Braintrusts und Thinktanks der Kriegs- und Nachkriegszeit, und diese Tradition wurde in den großen Softwareprojekten des militär-industriellen Komplexes langezeit beibehalten. Doch Ende der 60er Jahre wurde dieses Team-orientierte Software-

Manufactoring abrupt gestoppt, denn auch der finanziell und institutionell privilegierte "User" Militär geriet angesichts steigender Softwareausgaben unter stärkeren Kostendruck. Man forderte nun eine striktere Reglementierung des "system life cycle" im Hinblick auf "reuse", "compatiblity" und "transferabilty". In diesem Kontext setzte sich seit 1967 das keine Rekursionen zulassende Wasserfallmodell immer mehr durch. Das Leitbild "Software-Manufaktur" wurde damit im Contracting Sector endgültig von den Leitbildern der industriellen Softwareproduktion abgelöst. Die "component assembly", der "orderly process of software production" und ab 1971 bei SDC die "software factory" waren nun nicht mehr unverbindliche Metaphern, sondern strikte Leitlinien für die Software-Entwicklung.

Eine zweite Methodenrichtung am Übergang von der handwerklichen zur quasi-industriellen Software-Entwicklung ging im Laufe der 60er Jahre aus den Bemühungen großer Hardware- und Software-Produzenten hervor, um die wachsende Komplexität von Betriebssystemen und großen Anwendungsprogrammen für Flugreservierungssysteme und dergl. zu bewältigen. Hier kam es auch zu den spektakulärsten Zeitüberschreitungen, Fehlerhäufungen und Projektdesastern. Bereits 1961/62 tauchten in diesem Zusammenhang Klagen über eine "software crisis" auf. Die unbewältigte Komplexität großer Software-Systeme wurde daher auch im Corporate Sector des Software-Marktes Anlass für die Suche nach Methoden einer klareren Strukturierung von Programmsystemen und Entwicklungsprozessen. Man fand sie in Strategien der Hierarchisierung der Systeme, der Modularisierung und der Bildung von Abstraktionsebenen. Mit der stärkeren Gewichtung einer strikten Abgrenzung der Teilaufgaben und der vom Projektmanagement kontrollierten Arbeitsteilung kündigte sich der Übergang von den am Systems Engineering angelehnten, Management- und Team-orientierten Ansätzen der Software-Entwicklung zu einer am Industrial Engineering angelehnten Vorgehensweise an. Diese gab sich nicht mehr mit einer Phasenbildung zufrieden, die eher Leitbildcharakter hatte, und ebenso wenig mit dem bisherigen Kompromiss zwischen *horizontaler* modularer Aufgabenteilung und interdependenter Problemstruktur, der den Kommunikationsaufwand nicht entscheidend zu verringern vermochte. Man suchte vielmehr die Lösung in einer Arbeitsteilungsgerechten Systemstrukturierung, d. h. einer *vertikalen* Modularisierung und einer interferenzfreien Zerlegung der komplexen Programm- und Informationssysteme. James Martin wollte 1965 nach dem Muster des Austauschbaus in der Mechanik und der Komponentenstruktur der Elektrokonstruktion die Interdependenzen der Programmsegmente und die komplexe Interaktion zwischen den Programmierern durch eine "division into sub-systems" und eine Abschottung der Einheiten bereits vor Beginn des Entwicklungsprozesses ausschalten. Die

Engineering-Metapher ist damit drei Jahre vor der Garmisch-Konferenz bereits voll präsent, es fehlt nur noch der eigentliche Begriff "Software-Engineering".

Doch bereits Mitte der 60er Jahre wurden diese konsequenten Software-Engineering-Positionen von anderen Designauffassungen im IBM-Umfeld grundsätzlich infrage gestellt. So reichten vor allem für Frederick Brooks *Dekompositionsmethoden* zur Komplexitätsbewältigung in Softwaresystemen nicht aus. Da es in diesen darum gehe, heterogene Interessen der beteiligten Stakeholder auszugleichen und divergierende Designanforderungen nutzer- und nutzungsgerecht zu einem konsistenten und möglichst kostengünstigen Systementwurf zu integrieren, beruhe auch die Software-Entwicklung wesentlich auf *Kompositionsmethoden*. Brooks brachte deshalb 1965 in Anlehnung an den von ihm selber schon 1960/61 geprägten Begriff "computer architecture" die Bezeichnung "software architecture" auf. Mit dem diesem Begriff verzichtete er nicht auf strukturierende Methoden, aber er ergänzte sie vor allem in den strukturbildenden Designphasen um integrierende Designmethoden, die die Wünsche der "user", sowie die Qualität und Konsistenz des Designs in das Zentrum stellten. Weitere Äußerungen von IBM-Vertretern in dieser Zeit belegen, dass Brooks mit seiner Kritik an szientistischen Auffassungen des Software-Designs keineswegs alleine stand. Auf den beiden NATO-Konferenzen in Garmisch und Rom bildete sich gerade in diesem Umfeld eine stille Opposition der Praktiker gegen den vorherrschenden Theorieanspruch der akademischen Software-Engineering-Protagonisten, die die Softwarekrise durch eine auf Verwissenschaftlichung beruhende Rationalisierung und Industrialisierung der traditionellen Software-Produktion zu bewältigen hofften.

Im Laufe der 70er Jahre verschob sich die Bedeutung des Software-Architektur-Begriffes langsam von der Kompositions- zur Dekompositionslehre. Nach 1975 wurden dann Begriff und Konzept der Software-Architecture von der Leitmetapher Software Engineering für nahezu zwei Jahrzehnte völlig an den Rand gedrängt. Erst in der ersten Hälfte der 90er Jahre setzte sich wieder die Einsicht durch, dass die Softwareentwicklung neben den oder gar statt der Engineering-Methoden architektonischer Gestaltung bedürfe. Das wiedererstandene Konzept der Software Architecture, das sich rhetorisch teilweise eng an Frederick Brooks anlehnte, wurde schnell zum Sammelbecken aller ungelösten Probleme und nicht erfüllten Versprechen des Software-Engineering. Dabei wird unter Software Architecture sowohl eine *systematische Vollendung des Software-Engineering* durch Erschließung, Durchdringung und Systematisierung bislang theoretisch ausgesparter informeller Designbereiche gesehen als auch eine *eigene Teildisziplin neben dem Software-Engineering* verstanden, die für weniger formalisierbare

Design- und Management-Aufgaben zuständig ist oder sie wird gar als eine *Alternative zum bisher glücklosen Software-Engineering* aufgefasst, die die Komplexitätsbewältigung in der Software-Konstruktion durch eine Erneuerung der künstlerisch-integrativen Designphilosophie des Brookschen Architektur-Konzeptes anstrebt. So zeigt auch die Geschichte der Software-Konstruktionsmethoden

- 1. dass die Leitbild-Fokussierung der Software-Konstruktionsmethodik ein z. T. recht ähnliches Entwicklungsmuster aufweist wie die Mechanik-, Elektro-, Computer- und Netzwerkkonstruktion, nämlich dass sich *Kompositionslehren*, die der Designkomplexität voll Rechnung tragen, mit *Dekompositionslehren* ablösen, die vorrangig auf Standardisierung, Prozessrationalisierung und Reduktion von Designkomplexität zielen.

- 2. dass auch die Software-Entwickler des Contractor-Sektors zunächst einen schnellen Einstieg in die ingenieurmäßige industrielle Software-Entwicklung erhofften, aber sehr bald die Besonderheiten des Produktes Software erkannten, die auch in arbeitsteiligen Entwicklungsprozessen einen hohen Bedarf an Kooperation und Kommunikation erforderlich machten.

- 3. dass sich der Verzicht auf eine rein hierarchische Ablaufsteuerung des Software-Entwicklungsprozesses zugunsten einer arbeitsteiligen Kooperation in den Modellierungswerkzeugen niederschlug, so dass in dieser Periode Phasenmodelle entstanden, die sich deutlich von den rigiden Wasserfall-Modellen der 70er und 80er Jahre unterschieden,

- 4. dass sich Software-Enginering bzw. -Architektur wie die Computerarchitektur als eine konstruktive Technikdisziplin erweist, deren wandelnde Akteurs- und Nutzungskontexte ständige Verschiebungen und Neuaushandlungen der Designmerkmale bewirken und dass ein an naturwissenschaftsnahen Technikwissenschaften ausgerichtetes Verwissenschaftlichungsmodell daher hier nur begrenzte Geltung hat.

- 5. dass in den Konstruktionslehren des Mechanik-, Elektro-, Computer- und Software-Bereiches gleichermaßen lineare Fortschrittskonzepte, die einen verbindlichen Entwicklungsgang von der „Kunst“ über die Erfahrungswissenschaft zu einer theoriebasierten ‘exakten’ ingenieurwissenschaftlichen Disziplin postulieren, unangemessen und für das in der Praxis notwendige Zusammenspiel von Verwissenschaftlichung und Designorientierung hinderlich sind.